Carte de compilation des diagrammes de décision ordonné a valeurs réelles

Hélène Fargier, Pierre Marquis, Alexandre Niveau, Nicolas Schmidt

Carte de compilation des diagrammes de décision ordonnés à valeurs réelles *

Hélène Fargier[1] **Pierre Marquis**[2] **Alexandre Niveau**[3] **Nicolas Schmidt**[1,2]

[1] IRIT-CNRS, Univ. Paul Sabatier, Toulouse, France
[2] CRIL-CNRS, Univ. Artois, Lens, France
[3] GREYC-CNRS, Univ. Caen, France
fargier@irit.fr {marquis,schmidt}@cril.fr alexandre.niveau@unicaen.fr

Résumé

Les diagrammes de décision valués (VDDs) sont des structures de données représentant des fonctions à valeurs réelles positives. Ces structures sont utiles pour la compilation de fonctions de coût ou d'utilité, ou encore de distributions de probabilités. Si la complexité de certaines requêtes (comme l'optimisation) et de certaines transformations (comme le conditionnement) sur de tels langages est bien connue, il reste de nombreuses requêtes et transformations importantes dont la complexité n'a pas encore été identifiée ; figurent parmi elles différents types de coupes, marginalisations, ou encore combinaisons. En établissant une carte de compilation des diagrammes de décision ordonnés à valeurs réelles, cet article contribue à combler ce manque. Nos résultats montrent que beaucoup de tâches difficiles à partir de CSPs valués sont traitables à partir de VDDs.

Abstract

Valued decision diagrams (VDDs) are data structures that represent functions mapping variable-value assignments to non-negative real numbers. They prove useful to compile cost functions, utility functions, or probability distributions. While the complexity of some queries (notably optimization) and transformations (notably conditioning) on VDD languages has been known for some time, there remain many significant queries and transformations, such as the various kinds of cuts, marginalizations, and combinations, the complexity of which has not been identified so far. This paper contributes to filling this gap and completing previous results about the time and space efficiency of VDD languages, thus leading to a knowledge compilation map for real-valued functions. Our results show that many tasks that are hard on valued CSPs are actually tractable on VDDs.

*Cet article est la version française de « A Knowledge Compilation Map for Ordered Real-Valued Decision Diagrams », à paraître dans les actes de la conférence AAAI'2014 [11].

1 Introduction

Les diagrammes de décision valués sont des structures de données représentant des fonctions à variables multiples, et à valeur dans un ensemble $\mathcal{V}$ de « valuations » (généralement un sous-ensemble de $\mathbb{R}^+$). Ces fonctions correspondent souvent à des fonctions de coût ou d'utilité, ou à des distributions de probabilités, qui sont largement utilisés en IA. Parmi les différentes tâches importantes lorsque l'on utilise ce genre de fonctions, on retrouve la requête d'*optimisation* : trouver une affectation donnant la valuation optimale, ou la valeur d'une variable donnée pouvant conduire à une affectation optimale, etc. Une telle requête est particulièrement intéressante combinée avec la transformation de *conditionnement*, qui donne une représentation restreinte de la fonction, dans laquelle certaines variables ont été affectées. La combinaison de ces deux tâches permet par exemple de résoudre des problèmes comme la recherche du diagnostic le plus probable considérant un ensemble de symptômes, ou de la voiture la moins chère d'une gamme qui soit munie d'un klaxon cucaracha.

Plusieurs autres structures de données ont été définies pour représenter ce genre de fonctions à variables multiples, les plus connues étant les CSP valués [19], les GAI nets [4], et les réseaux bayésiens [17]. Cependant ces langages ne sont pas adaptés aux requêtes susmentionnées lorsque des garanties de temps de réponse sont imposées (comme c'est le cas dans les applications Web) : l'optimisation est en effet NP-difficile pour les CSPs, GAI-nets et réseaux bayésiens.

Les VDDs n'ayant pas cet inconvénient, plusieurs familles de VDDs ont été définies et étudiées durant

ces vingt dernières années. On s'intéressera ici notamment au langage ADD (*algebraic decision diagrams*) [5], au langage AADD (*affine algebraic decision diagrams*) [21, 18], et au langage SLDD (*semiring-labeled decision diagrams*) [22]. En réalité, SLDD est une famille de langages $\texttt{SLDD}_\otimes$, paramétrée par un opérateur $\otimes$. On retrouve en particulier $\texttt{SLDD}_+$ (équivalent au langage des EVBDDs [16, 15, 1]) lorsque $\otimes$ représente l'opérateur $+$, et $\texttt{SLDD}_\times$ lorsque $\otimes$ représente l'opérateur $\times$.

Toutefois, de nombreuses requêtes et transformations ne découlent pas du conditionnement et de l'optimisation. Considérons par exemple la requête suivante : « lister toutes les voitures bon marché (par exemple en-dessous de 10 000 euros) de telle gamme, d'une capacité d'au moins 8 passagers ». Cela demande de se concentrer sur les voitures bon marché, ce qui ne relève ni de l'optimisation ni du conditionnement. De même, certaines transformations, telles la projection sur une variable, ou son complémentaire, l'élimination de variables, qui ont une grande importance théorique et pratique (par exemple pour résoudre le problème « *posterior marginal* » (PM) dans les réseaux bayésiens), ne peuvent pas être réduites à l'optimisation et au conditionnement. C'est également le cas pour les transformations de combinaison (aussi appelées opérations « *apply* »), qui consistent, étant donné des représentations de fonctions f et g, à calculer une représentation de la fonction $f \odot g$, avec $\odot$ un opérateur associatif et commutatif sur $\mathcal{V}$ (par exemple l'addition ou la multiplication, lorsque $\mathcal{V} = \mathbb{R}^+$). Ces transformations sont très importantes, ne serait-ce que pour permettre la construction incrémentale de représentations par des algorithmes de type ascendant.

La carte de compilation [8] identifie la complexité des requêtes et transformations sur plusieurs langages propositionnels, ainsi que la compacité relative de ces langages. Cependant, elle ne concerne que le cas spécifique des fonctions booléennes ; et même si cette carte a été étendue dans de nombreuses directions, le cas des VDDs n'a pas encore été très développé. Il a été montré que le langage AADD est strictement plus compact que le langage ADD [18], les travaux sur la compacité ont ensuite été étendus [10], montrant que le langage AADD est strictement plus compact que les langages $\texttt{SLDD}_+$ et $\texttt{SLDD}_\times$, eux-mêmes plus compacts que le langage ADD. Pour compléter cette carte, il reste à déterminer l'ensemble des requêtes et transformations d'intérêt que chacun de ces langages satisfait. En d'autres termes, pour chaque requête et transformation, et pour chacun des langages ADD, $\texttt{SLDD}_+$, $\texttt{SLDD}_\times$ et AADD, il s'agit de mettre en évidence un algorithme en temps polynomial effectuant cette opération, ou de prouver qu'un tel algorithme n'existe pas à moins que $\mathsf{P} = \mathsf{NP}$.

C'est le but principal de cet article, qui est organisé comme suit La section suivante permet au lecteur de faire connaissance avec la famille des langages de VDDs, et la suivante présente les requêtes et transformations étudiées dans la carte de compilation. Nous exposons ensuite les résultats de complexité obtenus [1], établissant pour chaque langage étudié et chaque requête et transformation, si le langage satisfait ou pas la requête ou la transformation. Ils constituent une première avancée dans la construction d'une carte de compilation pour les fonctions à valeurs non booléennes.

2 Diagrammes de decision valués

Préliminaires. Soit un ensemble fini de variables $\mathcal{X} = \{x_1, \ldots, x_n\}$, chaque x_i prenant ses valeurs dans un domaine fini D_{x_i}, et soit un sous-ensemble $X \subseteq \mathcal{X}$; on note $\vec{x} = \{ \langle x_i, d_i \rangle \mid x_i \in X, d_i \in D_{x_i} \}$ une affectation des variables de X, et D_X l'ensemble de toutes ces affectations (le produit cartésien des domaines des variables de X). La concaténation de deux affectations $\vec{x}$ et $\vec{y}$ de deux ensembles disjoints de variables X et Y est une affectation de $X \cup Y$ notée $\vec{x} \cdot \vec{y}$.

On considère des fonctions f portant sur des variables d'un sous-ensemble $\mathrm{Scope}(f) \subseteq \mathcal{X}$, et prenant leurs valeurs dans $\mathcal{V}$ (dans cet article, on prendra souvent $\mathcal{V} = \mathbb{R}^+$). Le domaine de f est noté $D_f = D_{\mathrm{Scope}(f)}$. Pour tout $Z \subseteq \mathrm{Scope}(f)$, on note $f_{\vec{z}}$ la *restriction* (ou *conditionnement sémantique*) de f par $\vec{z}$, c'est-à-dire la fonction sur $\mathrm{Scope}(f) \setminus Z$ telle que pour tout $\vec{x} \in D_{\mathrm{Scope}(f)\setminus Z}$, $f_{\vec{z}}(\vec{x}) = f(\vec{z} \cdot \vec{x})$.

Soit un opérateur binaire $\odot$ sur $\mathcal{V}$, et deux fonctions f et g portant sur un même ensemble de variables ; $f \odot g$ est la fonction définie par $f \odot g\ (\vec{x}) = f(\vec{x}) \odot g(\vec{x})$. La $\odot$-projection de f sur $Z \subseteq \mathrm{Scope}(f)$ est définie par $f^{\odot,Z}(\vec{x}) = \bigodot_{\vec{y} \in D_{\mathrm{Scope}(f)\setminus Z}} f_{\vec{y}}(\vec{x})$.

En abusant quelque peu des notations, si X et Y sont deux ensembles de variables disjoints, $\mathrm{Scope}(f) = X$, et $\vec{x} \cdot \vec{y}$ est une affectation de $X \cup Y$, alors on suppose que $f(\vec{x} \cdot \vec{y}) = f(\vec{x})$. En pratique, on considère souvent des affectations complètes $\vec{x} \in D_{\mathcal{X}}$.

Soit $\succeq$ une relation binaire réflexive et transitive sur $\mathcal{V}$ (c'est-à-dire un préordre), dont on note $\sim$ la partie symétrique et $\succ$ la partie asymétrique. On définit les ensembles de « coupes » suivants :

- $CUT^{\max}(f) = \{ \vec{x}^* \mid \forall \vec{x}, \neg(f(\vec{x}) \succ f(\vec{x}^*)) \}$;
- $CUT^{\min}(f) = \{ \vec{x}^* \mid \forall \vec{x}, \neg(f(\vec{x}) \prec f(\vec{x}^*)) \}$;
- $CUT^{\succeq\gamma}(f) = \{ \vec{x} \mid f(\vec{x}) \succeq \gamma \}$;
- $CUT^{\preceq\gamma}(f) = \{ \vec{x} \mid f(\vec{x}) \preceq \gamma \}$;
- $CUT^{\sim\gamma}(f) = \{ \vec{x} \mid f(\vec{x}) \sim \gamma \}$;

1. Pour des raisons de place, les preuves ne sont pas développées ici ; une version complète (incluant les preuves) de l'article AAAI original peut être trouvée à l'adresse <url: https://niveau.users.greyc.fr/pub/AAAI14_FMNS.pdf>.

Par exemple, si l'optimisation correspond à la minimisation, $CUT^{\min}$ est l'ensemble des solutions optimales (les voitures les moins chères) ; $CUT^{\preceq\gamma}$ est l'ensemble des solutions satisfaisant la condition de la coupe (les voitures bon marché, celles dont le prix est inférieur à γ).

Un langage de représentation sur $\mathcal{X}$ à valeurs dans $\mathcal{V}$ est un ensemble de structures de données, équipé d'une fonction d'interprétation qui associe à chacune une fonction $f\colon D_{\mathcal{X}} \to \mathcal{V}$. Cette fonction est appelée l'*interprétation* de la structure de données, et la structure de données est la *représentation* de cette fonction.

Définition 2.1 (langage de représentation ; inspirée de [13])**.** Etant donné un ensemble de valuations $\mathcal{V}$, un *langage de représentation* $\mathtt{L}$ sur $\mathcal{X}$ à valeurs dans $\mathcal{V}$ est un quadruplet $\langle C_{\mathtt{L}}, \mathrm{Var}_{\mathtt{L}}, f^{\mathtt{L}}, s_{\mathtt{L}}\rangle$, où :

- $C_{\mathtt{L}}$ est un ensemble de structures de données (également appelées $\mathtt{L}$-représentations ou simplement « formules »),
- $\mathrm{Var}_{\mathtt{L}}\colon C_{\mathtt{L}} \to 2^{\mathcal{X}}$ est une fonction qui associe à chaque $\mathtt{L}$-représentation le sous-ensemble de $\mathcal{X}$ dont elle dépend,
- $f^{\mathtt{L}}$ est une fonction d'interprétation associant à chaque $\mathtt{L}$-représentation α une fonction $f^{\mathtt{L}}_{\alpha}$ depuis l'ensemble de toutes les affectations de $\mathrm{Var}_{\mathtt{L}}(\alpha)$ vers $\mathcal{V}$,
- $s_{\mathtt{L}}$ est une fonction de taille, de $C_{\mathtt{L}}$ dans $\mathbb{N}$, qui donne la taille de chacune des $\mathtt{L}$-représentations.

Diagrammes de décision valués. Dans la suite, nous considérons des langages de représentation basés sur des structures de données appelées *diagrammes de décision valués*. De tels diagrammes permettent de représenter des fonctions à valeurs dans $\mathcal{V}$ en autorisant les nœuds et les arcs à porter des valeurs, prises dans un ensemble $\mathcal{E}$ (généralement $\mathcal{E} = \mathcal{V}$, mais ce n'est pas toujours le cas, comme nous le verrons plus tard).

Définition 2.2 (diagramme de décision valué)**.** Un *diagramme de décision valué* (VDD) sur $\mathcal{X}$ et $\mathcal{E}$ est un multi-graphe acyclique orienté, fini, et comportant une racine unique, dans lequel chaque nœud N est étiqueté par une variable $x \in \mathcal{X}$ et possède un ensemble $\mathrm{Out}(N)$ de $|D_x|$ arcs sortants, chaque arc $a \in \mathrm{Out}(N)$ étant étiqueté par une valeur distincte $v(a) \in D_x$. Les arcs comme les nœuds peuvent être étiquetés par des éléments de $\mathcal{E}$; on note $\varphi(a)$ (resp. $\varphi(N)$) la valuation de l'arc a (resp. du nœud N). La taille d'un diagramme de décision α, notée $|\alpha|$, est la taille du graphe (son nombre de nœuds et d'arcs) plus la taille de l'ensemble des valuations portées par les arcs et les nœuds.

Nous supposons que tous les diagrammes de décision considérés sont *ordonnés*, i.e. que pour tout VDD, un ordre strict et total $\rhd$ sur $\mathcal{X}$ est choisi, et quel que soit le chemin pris de la racine à un nœud terminal, les variables étiquetant les nœuds sont rencontrés dans l'ordre $\rhd$ (chaque variable ne peut donc apparaître qu'une seule fois dans un même chemin). Un chemin de la racine à un nœud terminal représente une affectation (partielle) de $\mathcal{X}$; la structure des diagrammes de décision est *déterministe*, ce qui signifie qu'à une affectation donnée ne correspond au plus qu'un seul chemin dans α, noté $p_\alpha(\vec{x})$.

Un VDD α est considéré comme *réduit* s'il ne comporte aucun couple de nœuds isomorphes (distincts) [2]. Un système de cache peut être utilisé pour détecter et fusionner ces nœuds isomorphes à la volée, ce qui nous permet de considérer implicitement les diagrammes comme étant réduits.

Langages de la famille des VDDs. On s'intéresse à plusieurs langages de la famille des VDDs, à savoir `ADD`, `SLDD` et `AADD`. Chacun d'entre eux impose des restrictions sur les diagrammes admissibles (par exemple, dans le langage `ADD` seuls les nœuds terminaux portent des valuations φ), et définit comment ils sont interprétés. De ce fait, les langages `ADD`, `SLDD` et `AADD` diffèrent à la fois syntaxiquement (par la façon dont les arcs et les nœuds sont étiquetés) et sémantiquement (par la façon dont les formules sont interprétées). La figure 1 présente des représentations d'une même fonction dans chacun de ces langages, dont nous allons maintenant rappeler la définition.

Définition 2.3 (`ADD`)**.** Le langage `ADD` est le quadruplet $\langle C_{\mathtt{ADD}}, \mathrm{Var}_{\mathtt{ADD}}, f^{\mathtt{ADD}}, s_{\mathtt{ADD}}\rangle$, avec $C_{\mathtt{ADD}}$ l'ensemble des VDDs ordonnés sur $\mathcal{X}$ dont chaque nœud terminal est étiqueté par un élément de $\mathcal{E} = \mathcal{V}$ (en général, $\mathcal{E} = \mathcal{V} = \mathbb{R}^+$) et dont les arcs ne sont pas étiquetés, et $f^{\mathtt{ADD}}$ définie comme suit, pour chaque formule α et chaque affectation $\vec{x}$:

- si α est un nœud terminal, alors $f^{\mathtt{ADD}}_{\alpha}(\vec{x}) = \varphi(\alpha)$,
- sinon, en notant N la racine de α, $x \in \mathcal{X}$ sa variable, $d \in D_x$ la valeur telle que $\langle x, d\rangle \in \vec{x}$, $a = \langle N, M\rangle$ l'arc tel que $v(a) = d$, et β la formule `ADD` ayant pour racine le nœud M de α, alors $f^{\mathtt{ADD}}_{\alpha}(\vec{x}) = f^{\mathtt{ADD}}_{\beta}(\vec{x})$.

Dans le cadre `AADD` de Sanner & McAllester [18], le co-domaine de la fonction représentée est $\mathcal{V} = \mathbb{R}^+$. Un seul nœud terminal est autorisé, et chaque arc est étiqueté par un couple de valeurs de $\mathbb{R}^+$ (c'est-à-dire que $\mathcal{E} = \mathbb{R}^+ \times \mathbb{R}^+ \neq \mathcal{V}$).

2. Des nœuds N et M sont dits isomorphes s'ils sont étiquetés par la même variable x, portent (le cas échéant) le même φ-label, et qu'il existe une bijection B de $\mathrm{Out}(N)$ dans $\mathrm{Out}(M)$ telle que $\forall a \in \mathrm{Out}(N)$, a et $B(a)$ ont le même nœud terminal, partagent la même valeur de D_x, et $\varphi(a) = \varphi(B(a))$.

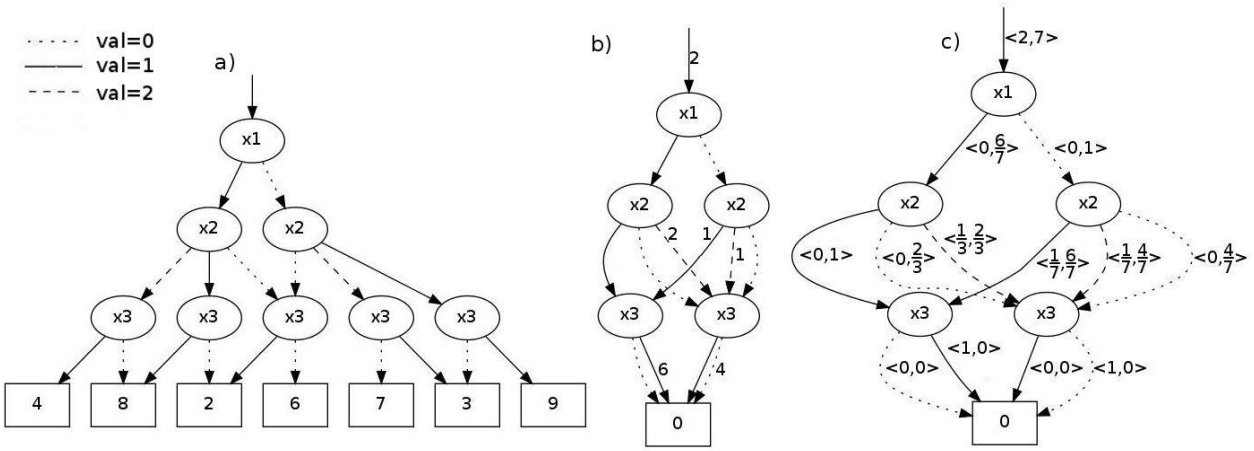

FIGURE 1 – Exemple de VDDs dans trois langages différents, ADD (a), $\texttt{SLDD}_+$ (b), et AADD (c), tous trois représentant la même fonction sur les variables $\{x_1, x_2, x_3\}$ avec $D_{x_1} = D_{x_3} = \{0, 1\}$ et $D_{x_2} = \{0, 1, 2\}$.

Définition 2.4 (AADD)**.** Le langage AADD est le quadruplet $\langle C_{\texttt{AADD}}, \text{Var}_{\texttt{AADD}}, f^{\texttt{AADD}}, s_{\texttt{AADD}}\rangle$, avec $C_{\texttt{AADD}}$ l'ensemble des VDDs ordonnés sur $\mathcal{X}$ ayant un unique nœud terminal, et dont les arcs sont étiquetés par un couple $\langle q, f\rangle$ dans $\mathbb{R}^+ \times \mathbb{R}^+$. A des fins de normalisation, la racine de α est également étiquetée par un couple $\langle q_0, f_0\rangle$ dans $\mathbb{R}^+ \times \mathbb{R}^+$, appelé *offset*. L'interprétation d'un tel VDD α est, pour chaque affectation $\vec{x}$, $f_\alpha^{\texttt{AADD}}(\vec{x}) = q_0 + (f_0 \times g_\alpha^{\texttt{AADD}}(\vec{x}))$, où $g_\alpha^{\texttt{AADD}}$ est définie comme suit :

- si α est le nœud terminal, alors $g_\alpha^{\texttt{AADD}}(\vec{x}) = 0$,
- sinon, en notant N la racine de α, $x \in \mathcal{X}$ sa variable, $d \in D_x$ la valeur telle que $\langle x, d\rangle \in \vec{x}$, $a = \langle N, M\rangle$ l'arc tel que $v(a) = d$, $\varphi(a) = \langle q_a, f_a\rangle$, et β la formule prenant racine au nœud M de α, alors $g_\alpha^{\texttt{AADD}}(\vec{x}) = q_a + (f_a \times g_\beta^{\texttt{AADD}}(\vec{x}))$.

Le langage AADD est muni d'une condition de normalisation qui rend toute AADD-représentation ordonnée (et réduite) canonique. La canonicité est importante car elle assure une représentation unique de chacune des sous-formules, ce qui est la clé pour efficacement reconnaître et fusionner les nœuds isomorphes.

Dans le cadre $\texttt{SLDD}_\otimes$ [22] [3], les arcs (mais pas les nœuds terminaux) sont étiquetés par des éléments de $\mathcal{E} = \mathcal{V}$, et $\langle \mathcal{E}, \otimes, 1_\otimes\rangle$ est un monoïde commutatif : $f_\alpha^{\texttt{SLDD}_\otimes}(\vec{x})$ est l'agrégation par $\otimes$ des étiquettes des arcs le long du chemin $p_\alpha(\vec{x})$.

Définition 2.5 ($\texttt{SLDD}_\otimes$)**.** Soit un monoïde commutatif $\langle \mathcal{E}, \otimes, 1_\otimes\rangle$; le langage $\texttt{SLDD}_\otimes$ est le quadruplet $\langle C_{\texttt{SLDD}_\otimes}, \text{Var}_{\texttt{SLDD}_\otimes}, f^{\texttt{SLDD}_\otimes}, s_{\texttt{SLDD}_\otimes}\rangle$, avec $C_{\texttt{SLDD}_\otimes}$ l'ensemble des VDDs ordonnés sur $\mathcal{X}$ ayant un unique nœud terminal N d'étiquette $\varphi(N) = 1_\otimes$, dont les arcs sont étiquetés avec des éléments de $\mathcal{E} = \mathcal{V}$, et dont la racine a un *offset* $\varphi_0 \in \mathcal{E}$. Pour chaque formule α et chaque affectation $\vec{x}$, $f_\alpha^{\texttt{SLDD}_\otimes}(\vec{x}) = \varphi_0 \otimes g_\alpha^{\texttt{SLDD}_\otimes}(\vec{x})$, où $g_\alpha^{\texttt{SLDD}_\otimes}(\vec{x})$ est définie comme suit :

- si α est le nœud terminal, alors $g_\alpha^{\texttt{SLDD}_\otimes}(\vec{x}) = 1_\otimes$,
- sinon, en notant N la racine de α, $x \in \mathcal{X}$ sa variable, $d \in D_x$ la valeur telle que $\langle x, d\rangle \in \vec{x}$, $a = \langle N, M\rangle$ l'arc tel que $v(a) = d$, et β la formule prenant racine au nœud M de α, alors $g_\alpha^{\texttt{SLDD}_\otimes}(\vec{x}) = \varphi(a) \otimes g_\beta^{\texttt{SLDD}_\otimes}(\vec{x})$.

Deux monoïdes sont particulièrement intéressants : $\langle \mathbb{R}^+, +, 0\rangle$ (le langage correspondant est noté $\texttt{SLDD}_+$) et $\langle \mathbb{R}^+, \times, 1\rangle$ (le langage correspondant est noté $\texttt{SLDD}_\times$). Les langages $\texttt{SLDD}_+$ et $\texttt{SLDD}_\times$ ont tous deux des conditions de normalisation qui assurent leur canonicité. De plus, toute formule de l'un de ces deux langages peut être traduite en temps linéaire vers le langage AADD .

Il en va de même pour le langage ADD, dont chaque formule peut être traduite en temps linéaire en une formule de chacun des trois autres langages étudiés ici ; un exemple est donné en figure 2 (pour plus de détails sur les traductions de formules d'un langage vers un autre, voir Fargier et al. [9]). Enfin, notons que si les variables sont booléennes, et $\mathcal{V} = \{0, 1\}$, toute formule de chacun des quatre langages considérés peut être traduite en un OBDD en temps linéaire, la réciproque étant trivialement vraie.

3. Notre définition de SLDD est un peu plus générale que l'originale, en ce que (i) nous ne considérons que des diagrammes ordonnés, et (ii) nous utilisons un monoïde commutatif au lieu d'un semi-anneau commutatif, le deuxième opérateur n'étant pas utilisé dans la définition de la structure de données [10].

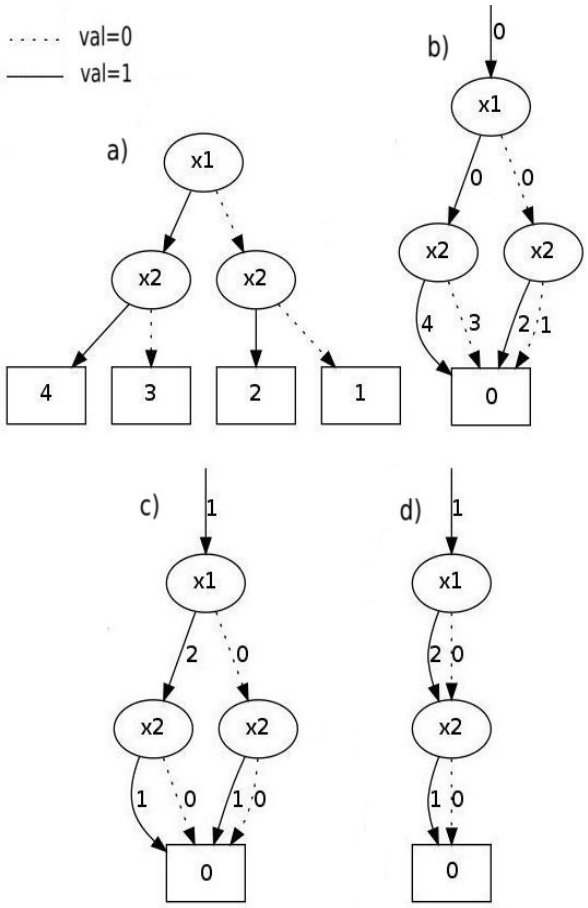

FIGURE 2 – Exemple de traduction d'une formule du langage ADD vers le langage SLDD_+ : (a) formule ADD de départ, (b) les poids sont remontés sur les arcs, (c) les nœuds x_2 puis x_1 sont normalisés, et (d) les nœuds isomorphes sont fusionnés.

3 Requêtes et transformations

Nous introduisons maintenant plusieurs requêtes et transformations importantes, que nous définissons dans le cadre général des langages de représentation à valeurs dans $\mathcal{V}$ — c'est-à-dire que ces requêtes et transformations sont bien définies même lorsque $\mathcal{V} \neq \mathbb{R}^+$.

Définition 3.1 (requêtes)**.** Soit L un langage de représentation sur $\mathcal{X}$ à valeurs dans un ensemble $\mathcal{V}$ totalement ordonné par une relation $\succeq$.

- L satisfait l'optimisation $\mathbf{OPT}_{\max}$ (resp. $\mathbf{OPT}_{\min}$) si et seulement s'il existe un algorithme en temps polynomial associant à toute formule α de L la valeur $\max_{\vec{x} \in D_f} f^{\mathtt{L}}_\alpha(\vec{x})$ (resp. $\min_{\vec{x} \in D_f} f^{\mathtt{L}}_\alpha(\vec{x})$).
- L satisfait l'équivalence **EQ** si et seulement s'il existe un algorithme en temps polynomial associant tout couple α, β de formules de L à 1 si $f^{\mathtt{L}}_\alpha = f^{\mathtt{L}}_\beta$, et à 0 sinon.
- L satisfait l'implication de formules **SE** si et seulement s'il existe un algorithme en temps polynomial associant tout couple α, β de formules de L à 1 si $\forall \vec{x}, f^{\mathtt{L}}_\alpha(\vec{x}) \succeq f^{\mathtt{L}}_\beta(\vec{x})$, et à 0 sinon.
- L satisfait la cohérence partielle supérieure (resp. inférieure, resp. égale) $\mathbf{CO}_{\succeq\gamma}$ (resp $\mathbf{CO}_{\preceq\gamma}$, resp. $\mathbf{CO}_{\sim\gamma}$) si et seulement s'il existe un algorithme en temps polynomial associant tout $\gamma \in \mathcal{V}$ et toute formule α de L à 1 si $\exists \vec{x}, f^{\mathtt{L}}_\alpha(\vec{x}) \succeq \gamma$ (resp. $f^{\mathtt{L}}_\alpha(\vec{x}) \preceq \gamma$, resp. $f^{\mathtt{L}}_\alpha(\vec{x}) \sim \gamma$), et à 0 sinon.
- L satisfait la validité partielle supérieure (resp. inférieure, resp. égale) $\mathbf{VA}_{\succeq\gamma}$ (resp $\mathbf{VA}_{\preceq\gamma}$, resp. $\mathbf{VA}_{\sim\gamma}$) si et seulement s'il existe un algorithme en temps polynomial associant tout $\gamma \in \mathcal{V}$ et toute formule α de L à 1 si $\forall \vec{x}, f^{\mathtt{L}}_\alpha(\vec{x}) \succeq \gamma$ (resp. $f^{\mathtt{L}}_\alpha(\vec{x}) \preceq \gamma$, resp. $f^{\mathtt{L}}_\alpha(\vec{x}) \sim \gamma$), et à 0 sinon.
- L satisfait l'énumération de max-modèles $\mathbf{ME}_{\max}$ si et seulement s'il existe un polynôme p et un algorithme associant à toute formule α de L l'ensemble des éléments de $CUT^{\max}(f^{\mathtt{L}}_\alpha)$ en temps $p(|\alpha|, |CUT^{\max}(f^{\mathtt{L}}_\alpha)|)$.
- L satisfait l'extraction de max-modèle $\mathbf{MX}_{\max}$ si et seulement s'il existe un algorithme en temps polynomial associant à toute formule α de L un élément de $CUT^{\max}(f^{\mathtt{L}}_\alpha)$.
- L satisfait le comptage de max-modèles $\mathbf{CT}_{\max}$ si et seulement s'il existe un algorithme en temps polynomial associant à toute formule α de L le nombre d'éléments de $CUT^{\max}(f^{\mathtt{L}}_\alpha)$.

Les requêtes $\mathbf{ME}_{\min}$, $\mathbf{ME}_{\succeq\gamma}$, $\mathbf{ME}_{\preceq\gamma}$ et $\mathbf{ME}_{\sim\gamma}$ (resp. $\mathbf{MX}_{\min}$, $\mathbf{MX}_{\succeq\gamma}$, $\mathbf{MX}_{\preceq\gamma}$ et $\mathbf{MX}_{\sim\gamma}$; resp. $\mathbf{CT}_{\min}$, $\mathbf{CT}_{\succeq\gamma}$, $\mathbf{CT}_{\preceq\gamma}$ et $\mathbf{CT}_{\sim\gamma}$) sont définies de la même façon que $\mathbf{ME}_{\max}$ (resp. $\mathbf{MX}_{\max}$; resp. $\mathbf{CT}_{\max}$), en utilisant respectivement les ensembles $CUT^{\min}(f^{\mathtt{L}}_\alpha)$, $CUT^{\succeq\gamma}(f^{\mathtt{L}}_\alpha)$, $CUT^{\preceq\gamma}(f^{\mathtt{L}}_\alpha)$ et $CUT^{\sim\gamma}(f^{\mathtt{L}}_\alpha)$ à la place de $CUT^{\max}(f^{\mathtt{L}}_\alpha)$.

Les familles de requêtes **MX** et **ME** sont cruciales pour le raisonnement bayésien et la configuration interactive. Elles capturent, par exemple, la demande d'une explication parmi les plus probables ($\mathbf{MX}_{\max}$), ou d'une des configurations les moins chères ($\mathbf{MX}_{\min}$). Dans ces applications, le comptage est également utile, comme pour caractériser le nombre de voitures « bon marché » ($\mathbf{CT}_{\preceq\gamma}$) ou le nombre de maladies suffisamment probables pour être considérées ($\mathbf{CT}_{\succeq\gamma}$). Les autres requêtes, telles la cohérence, la validité, l'équivalence ou l'implication, sont utiles à de nombreux problèmes de raisonnement, notamment car elles étendent aux bases de connaissances pondérées les requêtes analogues définies sur les formules booléennes en NNF.

Définition 3.2 (transformations)**.** Soit L un langage de représentation sur $\mathcal{X}$ à valeurs dans $\mathcal{V}$, et $\odot$ un opérateur binaire associatif et commutatif sur $\mathcal{V}$.

- L satisfait le conditionnement **CD** si et seulement s'il existe un algorithme en temps polynomial associant à toute formule α de L, tout $X \subseteq \mathcal{X}$, et tout $\vec{x} \in D_X$, une L-représentation de $f^{\mathtt{L}}_{\alpha,\vec{x}}$.
- L satisfait la $\odot$-combinaison bornée $\odot$**BC** si et seulement s'il existe un algorithme en temps polynomial associant à tout couple α, β de formules de L, une L-représentation de $f^{\mathtt{L}}_{\alpha} \odot f^{\mathtt{L}}_{\beta}$.
- L satisfait la $\odot$-combinaison $\odot$**C** si et seulement s'il existe un algorithme en temps polynomial associant à tout ensemble $\{\alpha_1, \ldots, \alpha_n\}$ de formules de L, une L-représentation de $\bigodot_{i=1}^{n} f^{\mathtt{L}}_{\alpha_i}$.
- L satisfait la $\odot$-élimination de variables $\odot$**Elim** (resp. d'une seule variable, **S**$\odot$**Elim**) si et seulement s'il existe un algorithme en temps polynomial associant à toute formule α de L et tout ensemble de variables $X \subseteq \mathcal{X}$ (resp. tout singleton $X \subseteq \mathcal{X}$) une L-représentation de $\bigodot_{\vec{x} \in D_X} f^{\mathtt{L}}_{\alpha,\vec{x}}$.
- L satisfait la $\odot$-élimination d'une variable bornée **SB**$\odot$**Elim** si et seulement s'il existe un polynôme p et un algorithme associant à toute formule α de L et toute variable $x \in \mathcal{X}$, une L-représentation de $\bigodot_{\vec{x} \in D_x} f^{\mathtt{L}}_{\alpha,\vec{x}}$ en temps $p(|\alpha|^{|D_x|})$.
- L satisfait la $\odot$-marginalisation de variable $\odot$**Marg** si et seulement s'il existe un algorithme en temps polynomial associant à toute formule α de L et toute variable $x \in \mathcal{X}$ une L-représentation de $\bigodot_{\vec{z} \in D_{\mathcal{X}\setminus\{x\}}} f^{\mathtt{L}}_{\alpha,\vec{z}}$.
- L satisfait la γ-coupe haute $\mathbf{CUT}_{\succeq\gamma}$, suivant un préordre $\succeq$ sur $\mathcal{V}$, si et seulement s'il existe un algorithme en temps polynomial associant à toute formule α de L et toutes valeurs $\gamma, a, b \in \mathcal{V}$ vérifiant $a \succ b$, une L-représentation de la fonction g définie par $g(\vec{x}) = a$ si $\vec{x} \in CUT^{\succeq\gamma}(f^{\mathtt{L}}_{\alpha})$, et $g(\vec{x}) = b$ sinon.

Les transformations $\mathbf{CUT}_{\preceq\gamma}$, $\mathbf{CUT}_{\sim\gamma}$, $\mathbf{CUT}_{\max}$ et $\mathbf{CUT}_{\min}$ sont définies de la même façon, en utilisant respectivement les ensembles $CUT^{\preceq\gamma}(f^{\mathtt{L}}_{\alpha})$, $CUT^{\sim\gamma}(f^{\mathtt{L}}_{\alpha})$, $CUT^{\max}(f^{\mathtt{L}}_{\alpha})$ et $CUT^{\min}(f^{\mathtt{L}}_{\alpha})$.

La transformation classique d'« oubli » de variables (*forgetting*) correspond à leur max-élimination. La marginalisation sur une variable est équivalente à l'élimination de toutes les variables, sauf une ; la +-marginalisation est typiquement importante dans le cadre des réseaux bayésiens, pour effectuer la *posterior marginal request* (voir par exemple Darwiche [7]), et dans les problèmes de configuration, la min-marginalisation dans un VDD représentant une fonction de prix revient à calculer le prix minimal associé à toute valeur possible de la variable en question (qui représente par exemple une option pour une voiture, voir Astesana et al. [3]). On trouve un autre exemple d'utilisation en planification stochastique, où l'algorithme d'itération de la valeur peut être a été implémenté par une suite de +-éliminations et de ×-combinaisons sur des formules ADD [14].

Pour finir, la famille des opérations de coupe représente la restriction d'une fonction aux affectations optimales (les voitures les moins chères, les explications les plus probables…) ou proches de l'optimum (les voitures à moins de γ euros, les maladies ayant une probabilité $> \gamma$). Pour qu'elle reste aussi générale que possible, nous avons défini cette famille d'opérations comme des transformations au sein d'un même langage, mais il est important de noter qu'en prenant $a = 1$ et $b = 0$ quand $\mathcal{V} = \mathbb{R}^+$, la coupe d'un VDD est en fait une formule du langage MDD [20], une extension directe d'OBDD aux variables non booléennes, qui satisfait à peu près les mêmes requêtes et transformations, comme **CO**, **CD**, **SE**, etc. [2].

4 Carte de compilation des VDDs ordonnés à valeurs dans $\mathbb{R}^+$

Nous pouvons à présent établir une carte de compilation des VDDs représentant des fonctions à valeur réelle ; on se focalise donc dans la suite de l'article sur le cas $\mathcal{V} = \mathbb{R}^+$, avec $\succeq$ l'ordre habituel $\geq$ sur $\mathbb{R}^+$, et $\odot \in \{\max, \min, +, \times\}$. Les langages considérés sont ADD, $\mathtt{SLDD}_{\times}$, $\mathtt{SLDD}_{+}$ et AADD ; nous laissons de côté $\mathtt{SLDD}_{\max}$ et $\mathtt{SLDD}_{\min}$, ces langages ayant été prouvés équivalents à ADD modulo une transformation polynomiale [10].

Commençons par les requêtes de base. Chacun des quatre langages examinés satisfait **CD** : conditionner une formule α par une affectation $\vec{x}$ peut être effectué en temps linéaire grâce à un algorithme adapté du cas OBDD. De plus, comme la représentation de toute fonction en une formule réduite (resp. réduite et normalisée) de ADD (resp. AADD, $\mathtt{SLDD}_{\times}$, $\mathtt{SLDD}_{+}$) est unique, et comme les procédures de réduction et normalisation sur chacun des quatre langages sont polynomiales, ils satisfont tous **EQ**. Enfin, **SE** est satisfaite par ADD, $\mathtt{SLDD}_{+}$ et $\mathtt{SLDD}_{\times}$, grâce à une combinaison de coupes, d'inversion des valuations sur les arcs et de traduction vers MDD. Décider si AADD satisfait **SE** reste en revanche un problème ouvert.

Proposition 4.1.

- ADD, $\mathtt{SLDD}_{+}$, $\mathtt{SLDD}_{\times}$ *et* AADD *satisfont* **EQ** *et* **CD**.
- ADD, $\mathtt{SLDD}_{+}$ *et* $\mathtt{SLDD}_{\times}$ *satisfont* **SE**.

Pour structurer les autres résultats, nous distinguons trois catégories d'opérations : celles liées à l'optimisation, qui sont traitables sur les VDDs ; celles liées à la coupe suivant un seuil γ, qui peuvent devenir difficiles ; et les transformations de combinaison et d'élimination de variables. Les résultats obtenus pour les deux premières catégories sont résumés dans la table 1, et ceux obtenus pour la troisième le sont dans la table 2.

Opérations liées à l'optimisation. Comme évoqué précédemment, $\mathbf{OPT}_{\max}$ et $\mathbf{OPT}_{\min}$ sont traitables pour les formules AADD [18] comme pour les formules SLDD [22], et leur satisfaction par ADD est évidente. Tous les résultats positifs de la table 1 sont liés aux faits que (i) les VDDs sont des graphes sans circuit, et (ii) l'agrégation des valuations φ est monotone sur les classes considérées. Dans de tels diagrammes, les chemins de valeur minimale (resp. maximale) peuvent être obtenus en temps polynomial, grâce à un algorithme de recherche de plus court (resp. plus long) chemin. En s'appuyant sur cette idée, on peut écrire une procédure en temps polynomial construisant une formule MDD qui représente les affectations optimales — ce qui implique la satisfaction de $\mathbf{CUT}_{\max}$ et $\mathbf{CUT}_{\min}$, dont on peut déduire celle des requêtes portant sur ces coupes.

Proposition 4.2. ADD, SLDD_+, $\text{SLDD}_\times$ *et* AADD *satisfont* $\mathbf{OPT}_{\max}$, $\mathbf{OPT}_{\min}$, $\mathbf{CUT}_{\max}$, $\mathbf{CUT}_{\min}$, $\mathbf{ME}_{\max}$, $\mathbf{ME}_{\min}$, $\mathbf{MX}_{\max}$, $\mathbf{MX}_{\min}$, $\mathbf{CT}_{\max}$ *et* $\mathbf{CT}_{\min}$.

Opérations liées aux γ-coupes. Les requêtes et transformations liées à l'optimisation pure sont donc faciles ; cependant, pour de nombreuses applications, l'optimisation seule ne suffit pas. Ainsi, un client voulant acheter une voiture ne cherche pas forcément la moins chère, mais un véhicule qui satisfait ses desiderata tout en étant relativement bon marché, c'est-à-dire, d'un coût inférieur à un certain seuil — d'où l'importance des γ-coupes.

Un résultat positif est qu'il suffit de comparer la valeur maximale (resp. minimale) de $f_\alpha^{\mathtt{L}}$ (qui peut s'obtenir en temps polynomial) à $\gamma \in \mathbb{R}^+$ pour décider s'il existe un $\vec{x}$ tel que $f_\alpha^{\mathtt{L}}(\vec{x}) \geq \gamma$ (resp. $\leq \gamma$). De même, décider de la γ-validité d'une formule α nécessite seulement de comparer γ à la valeur maximale (resp. minimale) de $f_\alpha^{\mathtt{L}}$.

Proposition 4.3. ADD, SLDD_+, $\text{SLDD}_\times$ *et* AADD *satisfont* $\mathbf{CO}_{\succeq\gamma}$, $\mathbf{VA}_{\succeq\gamma}$, $\mathbf{CO}_{\preceq\gamma}$, $\mathbf{VA}_{\preceq\gamma}$ *et* $\mathbf{VA}_{\sim\gamma}$.

Bien qu'il soit possible de vérifier en temps polynomial s'il existe une affectation conduisant à une valeur supérieure ou égale à γ, déterminer s'il existe une affectation conduisant *exactement* à γ est difficile sur SLDD_+, $\text{SLDD}_\times$ et AADD. La preuve s'appuie sur une réduction polynomiale depuis le problème de décision SUBSET SUM [12].

Proposition 4.4. SLDD_+, $\text{SLDD}_\times$ *et* AADD *ne satisfont pas* $\mathbf{CO}_{\sim\gamma}$, *sauf si* $\mathsf{P} = \mathsf{NP}$.

Il est clair que la satisfaction de $\mathbf{MX}_{\sim\gamma}$ ou $\mathbf{CT}_{\sim\gamma}$ est une condition suffisante à celle de $\mathbf{CO}_{\sim\gamma}$; cela est vrai en toute généralité, et pas seulement sur nos quatre langages à valeurs dans $\mathbb{R}^+$. De façon similaire, on peut montrer que $\mathbf{ME}_{\sim\gamma}$ est également une condition suffisante à $\mathbf{CO}_{\sim\gamma}$, et que $\mathbf{CUT}_{\sim\gamma}$ implique $\mathbf{MX}_{\sim\gamma}$ si l'une des requêtes $\mathbf{MX}$ est satisfaite.

Proposition 4.5. *Soit* L *un langage de représentation sur $\mathcal{X}$ à valeurs dans un ensemble $\mathcal{V}$ totalement ordonné par une relation $\succeq$.*

- *Si* L *satisfait* $\mathbf{CUT}_{\sim\gamma}$ *et l'une des requêtes* $\mathbf{MX}$, *alors il satisfait* $\mathbf{MX}_{\sim\gamma}$.
- *Si* L *satisfait* $\mathbf{MX}_{\sim\gamma}$, $\mathbf{CT}_{\sim\gamma}$ *ou* $\mathbf{ME}_{\sim\gamma}$, *alors il satisfait* $\mathbf{CO}_{\sim\gamma}$.

Corollaire 4.6. SLDD_+, $\text{SLDD}_\times$ *et* AADD *ne satisfont pas* $\mathbf{MX}_{\sim\gamma}$, $\mathbf{CUT}_{\sim\gamma}$, $\mathbf{CT}_{\sim\gamma}$ *ou* $\mathbf{ME}_{\sim\gamma}$, *sauf si* $\mathsf{P} = \mathsf{NP}$.

Ainsi, excepté $\mathbf{VA}_{\sim\gamma}$, les requêtes visant la coupe exacte d'une fonction représentée par un VDD à un seul nœud terminal sont difficiles. La situation est plus nuancée lorsque l'on s'intéresse aux coupes inférieure et supérieure.

Proposition 4.7. *Soit* $\mathtt{L} \in \{\text{SLDD}_+, \text{SLDD}_\times, \text{AADD}\}$.

- L *satisfait* $\mathbf{ME}_{\succeq\gamma}$, $\mathbf{ME}_{\preceq\gamma}$, $\mathbf{MX}_{\succeq\gamma}$ *et* $\mathbf{MX}_{\preceq\gamma}$.
- L *ne satisfait ni* $\mathbf{CT}_{\succeq\gamma}$ *ni* $\mathbf{CT}_{\preceq\gamma}$, *sauf si* $\mathsf{P} = \mathsf{NP}$.
- L *ne satisfait ni* $\mathbf{CUT}_{\succeq\gamma}$ *ni* $\mathbf{CUT}_{\preceq\gamma}$.

L'idée de la preuve du premier point est que depuis tout nœud d'une formule AADD normalisée, il existe un chemin vers le nœud terminal qui correspond à la valeur 1, et un autre qui correspond à la valeur 0. Un chemin de la racine à ce nœud lui fournit un *offset* (notons-le $\langle p, q\rangle$), calculé à partir de ses arcs. Il est ainsi possible de répondre à $\mathbf{ME}_{\succeq\gamma}$ (resp. $\mathbf{ME}_{\preceq\gamma}$) en énumérant tous les chemins, sans explorer les nœuds pour lesquels $p + q < \gamma$ (resp. $p > \gamma$).

La preuve du point suivant s'appuie sur le fait que si $\mathbf{CT}_{\succeq\gamma}$ était satisfaite, il serait possible de compter les affectations $\vec{x}$ vérifiant $f_\alpha(\vec{x}) \geq \gamma$, et également celles vérifiant $f_\alpha(\vec{x}) > \gamma$; $\mathbf{CT}_{\sim\gamma}$ serait donc satisfaite, or nous avons montré qu'elle est difficile (même principe pour $\mathbf{CT}_{\preceq\gamma}$).

Les transformations $\mathbf{CUT}_{\succeq\gamma}$ et $\mathbf{CUT}_{\preceq\gamma}$ ne sont pas satisfaites, et ce inconditionnellement (même si $\mathsf{P} = \mathsf{NP}$). Pour le cas de $\mathbf{CUT}_{\succeq\gamma}$ sur SLDD_+, la preuve utilise le fait que la fonction $f(\vec{y} \cdot \vec{z}) = \sum_{i=1}^{n} y_i \cdot$

Requête	ADD	SLDD$_+$	SLDD$_\times$	AADD	VCSP$_+$
CD	√	√	√	√	√
EQ	√	√	√	√	?
SE	√	√	√	?	○
OPT$_{\max}$ / **OPT**$_{\min}$	√	√	√	√	○
CT$_{\max}$ / **CT**$_{\min}$	√	√	√	√	○
ME$_{\max}$ / **ME**$_{\min}$	√	√	√	√	○
MX$_{\max}$ / **MX**$_{\min}$	√	√	√	√	○
CUT$_{\max}$ / **CUT**$_{\min}$	√	√	√	√	?
VA$_{\sim\gamma}$	√	√	√	√	?
VA$_{\succeq\gamma}$ / **VA**$_{\preceq\gamma}$	√	√	√	√	○
CO$_{\sim\gamma}$	√	○	○	○	○
CO$_{\succeq\gamma}$ / **CO**$_{\preceq\gamma}$	√	√	√	√	○
ME$_{\sim\gamma}$	√	○	○	○	○
ME$_{\succeq\gamma}$ / **ME**$_{\preceq\gamma}$	√	√	√	√	○
MX$_{\sim\gamma}$	√	○	○	○	○
MX$_{\succeq\gamma}$ / **MX**$_{\preceq\gamma}$	√	√	√	√	○
CUT$_{\sim\gamma}$	√	○	○	○	?
CUT$_{\succeq\gamma}$ / **CUT**$_{\preceq\gamma}$	√	•	•	•	?
CT$_{\sim\gamma}$	√	○	○	○	○
CT$_{\succeq\gamma}$ / **CT**$_{\preceq\gamma}$	√	○	○	○	○

TABLE 1 – Résultats sur les requêtes de base, l'optimisation et les γ-coupes ; √ signifie « satisfait », • signifie « ne satisfait pas », et ○ signifie « ne satisfait pas, sauf si $\mathsf{P} = \mathsf{NP}$ ». Les résultats pour les problèmes de satisfaction de contraintes valuées additives (VCSP$_+$) sont données à titre de comparaison.

Transformation	ADD	SLDD$_+$	SLDD$_\times$	AADD
max**BC** / min**BC**	√	•	•	•
+BC	√	√	•	•
×BC	√	•	√	•
max**C** / min**C**	•	•	•	•
+C / **×C**	•	•	•	•
max**Elim** / min**Elim**	•	•	•	•
+Elim / **×Elim**	•	•	•	•
Smax**Elim** / **S**min**Elim**	•	•	•	•
S+Elim / **S×Elim**	•	•	•	•
SBmax**Elim** / **SB**min**Elim**	√	•	•	•
SB+Elim	√	√	•	•
SB×Elim	√	•	√	•
max**Marg** / min**Marg**	√	√	√	√
+Marg	√	√	√	√
×Marg	√	?	√	?

TABLE 2 – Résultats sur les transformations (voir légende des symboles en table 1). ; √ signifie « satisfait », • signifie « ne satisfait pas », et ○ signifie « ne satisfait pas, sauf si $\mathsf{P} = \mathsf{NP}$ ».

$2^{n-i} + \sum_{i=1}^{n} z_i \cdot 2^{n-i}$ a une SLDD$_+$-représentation comportant seulement $2n+1$ nœuds, en utilisant l'ordre $y_1 \lhd \cdots \lhd y_n \lhd z_1 \lhd \cdots \lhd z_n$ sur les variables, alors qu'il n'existe aucune formule en OBDD$_\lhd$ de taille polynomiale représentant la fonction qui vaut 1 quand $f(\vec{y} \cdot \vec{z}) \geq 2^n$ et 0 sinon. Les autres preuves sont similaires, en adaptant la fonction f.

Ces résultats sur AADD et SLDD contrastent avec le cas du langage ADD, qui satisfait toutes les transformations **CUT** (les γ-coupes s'obtiennent par une procédure simple de fusion de nœuds terminaux), et donc également toutes les requêtes **MX**, **ME**, **CT** et **CO**.

Combinaison, élimination de variables, marginalisation. Aucun des langages considérés ne satisfait les transformations de combinaison ou d'élimination de variables dans leur version non bornée ; ce n'est pas surprenant si l'on observe que (i) la combinaison disjonctive non bornée ($\vee$**C**) et l'oubli (**FO**) ne sont pas satisfaits par OBDD ; (ii) toute formule d'OBDD peut être vue comme une formule d'ADD, et donc être traduite en temps polynomial en SLDD$_+$ (resp. SLDD$_\times$, AADD) ; et (iii) la disjonction de plusieurs formules en OBDD revient à couper leur somme (leur combinaison par +) à la valeur minimale (par construction, les contre-modèles de la disjonction sont les $\vec{x}$ tels que $\varphi(\vec{x}) = 0$, la formule résultante peut donc être vue comme une formule en OBDD dont la négation est équivalente à la disjonction de la formule de départ) or tous les langages considérés satisfont **CUT**$_{\min}$. Les autres preuves s'appuient sur des arguments similaires. Il est à noter que contrairement au cas d'OBDD, même la $\odot$-élimination d'une seule variable est difficile (l'idée est que la $\odot$-combinaison des deux formules peut s'obtenir en $\odot$-éliminant une variable supplémentaire).

Proposition 4.8. ADD, SLDD$_+$, SLDD$_\times$ *et* AADD *ne satisfont pas* $\odot$**C**, $\odot$**Elim** *ou* **S**$\odot$**Elim**, *quelle que soit l'opération* $\odot \in \{\max, \min, +, \times\}$.

La question de la combinaison bornée est plus délicate. Sanner & McAllester [18] ont proposé une extension de l'algorithme « *apply* » de Bryant [6], lequel est en temps polynomial sur OBDD pour les opérations AND et OR, pour combiner deux formules en AADD (par +, ×, max ou min) ; cet algorithme est directement adaptable aux autres langages de la famille des VDDs considérés dans cet article. Cependant, la complexité de cette procédure « *apply* » n'avait pas été identifiée formellement ; un des principaux résultats de cet article est que les combinaisons bornées sont difficiles sur AADD, ce qui implique que l'algorithme en question n'est pas en temps polynomial.

Proposition 4.9.

- SLDD$_+$, SLDD$_\times$ *et* AADD *ne satisfont pas* max**BC**, min**BC**, **SB**max**Elim** *ou* **SB**min**Elim**.
- SLDD$_\times$ *et* AADD *ne satisfont pas* **+BC**, *et donc pas* **SB+Elim**.
- SLDD$_+$ *et* AADD *ne satisfont pas* **×BC**, *et donc pas* **SB×Elim**.

La difficulté de max**BC** et de min**BC** vient directement de celle de **CUT**$_{\succeq\gamma}$ et **CUT**$_{\preceq\gamma}$. Celle de **×BC** se prouve en considérant les deux fonctions suivantes, qui portent sur des variables booléennes : $f(\vec{x}) = \sum_{i=0}^{n-1} x_i \cdot 2^i$ (représentation d'un entier par un vecteur de bits) et $g(\vec{x}) = 2^{n+1} - f(\vec{x})$. Chacune a une

$\texttt{SLDD}_+$-représentation (pour l'ordre $x_0 \lhd x_1 \lhd \cdots \lhd x_{n-1}$) comportant $n+1$ nœuds et $2n$ arcs ; or il est possible de montrer que la $\texttt{SLDD}_+$-représentation de $f \times g$ (pour le même ordre) contient un nombre exponentiel d'arcs au dernier niveau, même après normalisation. Cela montre que toute $\texttt{SLDD}_+$-représentation de $f \times g$ est de taille exponentielle. Les autres preuves procèdent de la même manière, avec des fonctions f et g adaptées.

Il y a en fait deux cas dans lesquels la combinaison bornée n'est pas difficile : lorsque le langage considéré est ADD, et lorsque l'opérateur de combinaison est aussi celui servant à agréger les valeurs des arcs dans le langage considéré (l'addition pour $\texttt{SLDD}_+$ et le produit pour $\texttt{SLDD}_\times$). Dans ces deux cas, éliminer une variable de domaine borné est également réalisable en temps polynomial — par définition de l'élimination de variable.

Proposition 4.10.

- $\texttt{SLDD}_+$ *satisfait* $+$**BC** *et* **SB**$+$**Elim**.
- $\texttt{SLDD}_\times$ *satisfait* $\times$**BC** *et* **SB**$\times$**Elim**.
- ADD *satisfait* $\odot$**BC** *et* **SB**$\odot$**Elim** *pour toute opération* $\odot \in \{\times, +, \min, \max\}$.

Pour finir, nous avons prouvé que tous les langages considérés satisfont la marginalisation suivant tous les opérateurs, à l'exception de la $\times$-marginalisation sur AADD et $\texttt{SLDD}_+$, dont la satisfaction reste à déterminer.

Proposition 4.11.

- ADD, $\texttt{SLDD}_+$, $\texttt{SLDD}_\times$ *et* AADD *satisfont* $\odot$**Marg**, *pour toute opération* $\odot \in \{+, \min, \max\}$.
- ADD *et* $\texttt{SLDD}_\times$ *satisfont* $\times$**Marg**.

5 Conclusion

Cet article a présenté une analyse de la complexité des langages de la famille des VDDs à valeurs réelles pour un ensemble de requêtes et de transformations utiles. Les requêtes liées à l'optimisation se révèlent faisables pour tous les langages considérés, et c'est également le cas de la marginalisation additive et de certaines requêtes liées aux coupes ; ces dernières sont généralement difficiles lorsque l'on cherche à atteindre exactement une valeur γ, et cette difficulté s'étend aux transformations liées. Un des principaux résultats est que les combinaisons, même bornées, sont difficiles sur les VDDs, ce qui implique qu'aucun algorithme « *apply* » ne peut être en temps polynomial sur les formules en AADD dans le cas général, même pour les opérations simples et « booléennes » que sont min et max. Lorsqu'il est nécessaire de pouvoir effectuer des combinaisons bornées par $+$ (resp. par $\times$) efficacement, le meilleur compromis temps/espace est offert par $\texttt{SLDD}_+$ (resp. $\texttt{SLDD}_\times$). Enfin, il s'avère que la complexité de diverses opérations liées à l'optimisation est meilleure pour les langages de la famille des VDDs que pour d'autres langages dédiés à la représentation de fonctions non booléennes comme $\texttt{VCSP}_+$; des résultats proches sont à attendre pour les GAI-nets ou les réseaux bayésiens, pour lesquels l'optimisation est également difficile.

Remerciements

Ces travaux ont été en partie soutenus par le projet BR4CP ANR-11-BS02-008 de l'Agence Nationale de la Recherche.

Références

[1] Jérôme Amilhastre, Hélène Fargier, and Pierre Marquis. Consistency restoration and explanations in dynamic CSPs: Application to configuration. *Artificial Intelligence*, 135(1–2):199–234, 2002.

[2] Jérôme Amilhastre, Hélène Fargier, Alexandre Niveau, and Cédric Pralet. Compiling CSPs: A complexity map of (non-deterministic) multivalued decision diagrams. In *Proceedings of ICTAI'12*, pages 1–8, 2012.

[3] Jean-Marc Astesana, Laurent Cosserat, and Hélène Fargier. Constraint-based vehicle configuration: A case study. In *Proceedings of ICTAI'10*, pages 68–75, 2010.

[4] Fahiem Bacchus and Adam J. Grove. Graphical models for preference and utility. In *Proceedings of UAI'95*, pages 3–10, 1995.

[5] R. Iris Bahar, Erica A. Frohm, Charles M. Gaona, Gary D. Hachtel, Enrico Macii, Abelardo Pardo, and Fabio Somenzi. Algebraic decision diagrams and their applications. In *Proceedings of ICCAD'93*, pages 188–191, 1993.

[6] Randal E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, 35:677–691, 1986.

[7] Adnan Darwiche. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press, 2009.

[8] Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research (JAIR)*, 17:229–264, 2002.

[9] Hélène Fargier, Pierre Marquis, and Nicolas Schmidt. Compacité pratique des diagrammes

de décision valués : normalisation, heuristiques et expérimentations. In *Actes des Journées Francophones de Programmation par Contraintes (JFPC 2013)*, pages 123–132, 2013.

[10] Hélène Fargier, Pierre Marquis, and Nicolas Schmidt. Semiring labelled decision diagrams, revisited: Canonicity and spatial efficiency issues. In *Proceedings of IJCAI'13*, pages 884–890, 2013.

[11] Hélène Fargier, Alexandre Niveau, Pierre Marquis, and Nicolas Schmidt. A knowledge compilation map for ordered real-valued decision diagrams. In *Proceedings of AAAI'2014*, 2014. Accepté pour publication.

[12] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[13] G. Gogic, H.A. Kautz, Ch.H. Papadimitriou, and B. Selman. The comparative linguistics of knowledge representation. In *Proceedings of IJCAI'95*, pages 862–869, 1995.

[14] Jesse Hoey, Robert St-Aubin, Alan J. Hu, and Craig Boutilier. SPUDD: Stochastic planning using decision diagrams. In *Proceedings of UAI'99*, pages 279–288, 1999.

[15] Yung-Te Lai, Massoud Pedram, and Sarma B. K. Vrudhula. Formal verification using edge-valued binary decision diagrams. *IEEE Transactions on Computers*, 45(2):247–255, 1996.

[16] Yung-Te Lai and Sarma Sastry. Edge-valued binary decision diagrams for multi-level hierarchical verification. In *Proceedings of DAC'92*, pages 608–613, 1992.

[17] Judea Pearl. *Probabilistic reasoning in intelligent systems – networks of plausible inference*. Morgan Kaufmann series in representation and reasoning. Morgan Kaufmann, 1989.

[18] Scott Sanner and David A. McAllester. Affine algebraic decision diagrams (AADDs) and their application to structured probabilistic inference. In *Proceedings of IJCAI'05*, pages 1384–1390, 2005.

[19] Thomas Schiex, Hélène Fargier, and Gérard Verfaillie. Valued constraint satisfaction problems: Hard and easy problems. In *Proceedings of IJCAI'95 (1)*, pages 631–639, 1995.

[20] Arvind Srinivasan, Timothy Kam, Sharad Malik, and Robert K. Brayton. Algorithms for discrete function manipulation. In *Proceedings of ICCAD'90*, pages 92–95, 1990.

[21] Paul Tafertshofer and Massoud Pedram. Factored edge-valued binary decision diagrams. *Formal Methods in System Design*, 10(2/3), 1997.

[22] Nic Wilson. Decision diagrams for the computation of semiring valuations. In *Proceedings of IJCAI'05*, pages 331–336, 2005.